11.7 ابزارهایی برای کار با لیست ها: بسیاری از نیاز های ساختار داده را می توان با نوع لیست داخلی برآورده کرد. اگر چه، گاهی اوقات نیاز به پیاده سازی های جایگزین با مصالحه های(trade-offs) مختلف کارایی است.

ماژول array یک شی array() را ارائه می کند که مشابه لیست است و تنها داده های همگن را به صورت فشرده تر ذخیره می کند. مثال زیر یک آرایه ای از اعداد را نشان می دهد که به جای ۱۶ بایت به ازای هر ورودی از لیست های معمول اشیای int پایتون(این روش معمول است)، به صورت اعداد دودویی بدون علامت دو بایتی ذخیره شده اند (کد نوشتاری "H").

```
>>> from array import array
>>> a = array('H', [4000, 10, 700, 22222])
>>> sum(a)
26932
>>> a[1:3]
array('H', [10, 700])
```

ماژول collections یک شی deque() را ارائه می کند که مشابه لیست است و عملیات append و pop را از سمت چپ سریعتر، اما جستجوها(lookups) را در میانه آهسته تر انجام می دهد. این اشیا برای پیاده سازی صف ها و جستجو های درختی اول سطح بسیار مناسب هستند.

```
>>> from collections import deque
>>> d = deque(["task1", "task2", "task3"])
>>> d.append("task4")
>>> print("Handling", d.popleft())
Handling task1
```

```
unsearched = deque([starting_node])
def breadth_first_search(unsearched):
    node = unsearched.popleft()
    for m in gen_moves(node):
        if is_goal(m):
            return m
        unsearched.append(m)
```

علاوه بر پیاده سازی های لیست جایگزین، کتابخانه، ابزارهای دیگری از جمله ماژول bisect را به همراه توابعی برای دستکاری لیست های ذخیره شده ، نیز ارائه می کند.

```
>>> import bisect
>>> scores = [(100, 'perl'), (200, 'tcl'), (400, 'lua'), (500, 'python')]
>>> bisect.insort(scores, (300, 'ruby'))
>>> scores
[(100, 'perl'), (200, 'tcl'), (300, 'ruby'), (400, 'lua'), (500, 'python')]
```

ماژول heapq توابعی را برای پیاده سازی پشته ها بر اساس لیست های معمولی ارائه می کند. پایین ترین ورودی مقدار، همیشه در مکان صفر نگهداری می شود. این برای برنامه های کاربردی که مکررا به کوچکترین عنصر دسترسی دارند اما نمی خواهند یک جستجوی کامل روی لیست اجرا کنند، مفید است.

```
>>> from heapq import heapify, heappop, heappush
```

```
>>> data = [1, 3, 5, 7, 9, 2, 4, 6, 8, 0]
>>> heapify(data)                      # rearrange the list into heap order
>>> heappush(data, -5)                 # add a new entry
>>> [heappop(data) for i in range(3)]  # fetch the three smallest entries
[-5, 0, 1]
```